Neural Language Models and Few Shot Learning for Systematic Requirements Processing in MDSE

Vincent Bertram
bertram@se-rwth.de
RWTH Aachen University
Aachen, Germany

Miriam Boß
boss@se-rwth.de
RWTH Aachen University
Aachen, Germany

Evgeny Kusmenko
kusmenko@se-rwth.de
RWTH Aachen University
Aachen, Germany

Imke Helene Nachmann
nachmann@se-rwth.de
RWTH Aachen University
Aachen, Germany

Bernhard Rumpe
rumpe@se-rwth.de
RWTH Aachen University
Aachen, Germany

Danilo Trotta
danilo.trotta@rwth-aachen.de
RWTH Aachen University
Aachen, Germany

Louis Wachtmeister
wachtmeister@se-rwth.de
RWTH Aachen University
Aachen, Germany

Abstract

Systems engineering, in particular in the automotive domain, needs to cope with the massively increasing numbers of requirements that arise during the development process. The language in which requirements are written is mostly informal and highly individual. This hinders automated processing of requirements as well as the linking of requirements to models. Introducing formal requirement notations in existing projects leads to the challenge of translating masses of requirements and the necessity of training for requirements engineers. In this paper, we derive domain-specific language constructs helping us to avoid ambiguities in requirements and increase the level of formality. The main contribution is the adoption and evaluation of few-shot learning with large pretrained language models for the automated translation of informal requirements to structured languages such as a requirement DSL.

CCS Concepts: • **Software and its engineering** → **Requirements analysis**; • **Computing methodologies** → **Machine translation**.

Keywords: model-driven requirements engineering, few-shot learning, natural language processing

SLE '22, December 06–07, 2022, Auckland, New Zealand

ACM ISBN 978-1-4503-9919-7/22/12…$15.00
https://doi.org/10.1145/3567512.3567534

ACM Reference Format:
Vincent Bertram, Miriam Boß, Evgeny Kusmenko, Imke Helene Nachmann, Bernhard Rumpe, Danilo Trotta, and Louis Wachtmeister. 2022. Neural Language Models and Few Shot Learning for Systematic Requirements Processing in MDSE. In *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering (SLE '22), December 06–07, 2022, Auckland, New Zealand.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3567512.3567534

1 Introduction

The engineering of complex Cyber-Physical Systems (CPSs) such as Advanced Driver Assistance Systems (ADASs) is a highly effortful task that faces many challenges [18, 22, 30]. One of them is the rising number of requirements that address stakeholders from heterogeneous domains. In systems engineering, and in particular the automotive domain, requirements are captured as documents that contain text mainly in natural language [23] often with additional information provided through pictures or Computer-Aided Design (CAD) models. Experts interpret these textual requirements to enter the design phase, and most often derive details of the implementation directly from them [14]. The ambiguity of natural language, in particular, when interpreted by experts from different backgrounds, as well as the increasing number of requirements may result in decreasing product quality, system failures which are currently detected at late development stages [11] and hinders the implementation of functional safety standards such ISO 26262. Furthermore, the document-based approach to requirements engineering prevents agile development where automated analyses and syntheses should enable early error detection and fast feedback for the developers.

What is needed are tools to process requirements systematically during all phases of the development cycle. An

approach to achieve this is Model-Driven Engineering (MDE) [14], which utilizes models as primary development artifacts. These models serve as documentation and communication basis for engineers, but also as input for analyses and syntheses, such as verification [19], test case [11], or code generation [10]. For instance, MDE can be applied to facilitate the design of Artificial Intelligence (AI)-based systems [2, 20, 21]. Approaches to introduce MDE in the automotive requirements engineering exist [24], but introducing MDE comes with initial costs for training the domain experts in modeling and translating many documents to models [7]. An advantage of using Domain Specific Languages (DSLs) rather than general purpose modeling languages such as the Unified Modeling Language (UML) is that their syntax and semantics [15] can be designed to be intuitive for the model users. As requirements are captured in natural language, we assume that a textual DSL that offers sentence structures and wording that is close to the current formulation of requirements increases intuitiveness of both usage and understanding of models in this DSL significantly. However, in addition to the DSL development costs and the DSL training, once the DSL is developed, the translation of old, unstructured requirements to models in the DSL can be a tremendous effort due to the high number of requirements, requiring time and modeling know-how from the translating developer.

In this paper, we analyze an open source set of automotive requirements for ADAS and Adaptive Light System (ALS) to understand where formulation inaccuracies occur and how targeted DSL constructs can help eliminate these inaccuracies and increase the level of formality and consistency in these requirements. **The main contribution of this paper is the application and evaluation of few-shot learning of large neural natural language models for the translation of given unstructured requirements to sentences incorporating the new formal DSL constructs.**

Such translation models can be used 1) during the introduction phase of a DSL to automatically translate existing or legacy natural language requirements into the new DSL syntax and 2) to correct natural language inputs in a smart editor when a requirement engineer writes a new requirement as natural text. With this automation supported by the fact that few-shot learning requires only a handful of translation examples to learn a given translation task **our aim is to facilitate the introduction of highly specialized requirement DSLs, e.g. targeting a single department of a company using specific wording or even a single project.** Further technical details and examples are given in the accompanying technical report [4].

2 Preliminaries

Our approach for text-to-DSL translation of requirements relies heavily on large transformer-based neural language models [31]. For the automatic translation from natural language requirements to the DSL, we utilize a derivative of Generative Pre-trained Transformer (GPT) [28], which is tailored towards text generation. GPT is a transformer-based decoder-only language model that employs a semi-supervised learning approach [28]. The authors showed that generative unsupervised pretraining on unlabeled data, where, given a sequence of words, the network is supposed to learn to predict the next word with the highest likelihood, and subsequent supervised fine-tuning of the pretrained parameters for a specific downstream task outperformed discriminatively trained models. GPT language models have evolved over the last few years and various variants exist.

In [6], the authors show the few-shot learning capabilities of GPT-3. In few-shot learning, the model is given a *support set* consisting of a very small number of training examples demonstrating how to solve a new task as part of the model's input. No weight updates are necessary, i.e. no classical training is performed. The support set is input into the model as part of the query. Based on this context, the language model can then solve the new task for a new input. Few-shot learning enables targeted training for very specific tasks, making it particularly interesting for requirements engineering, a discipline heavily relying on natural language and where training data is often scarce.

For the automatic translation of natural language requirements to a model in the DSL we mainly rely on GPT-J-6B [32, 33], an open-source language model based on the 6.7B GPT-3 [6] network and its hyperparameters. According to the authors, its performance is almost on par with the 6.7B GPT-3 network and it is the best-performing publicly available transformer language model in terms of zero-shot performance on various down-streaming tasks[1].

The method presented in this paper is evaluated on a publicly available dataset published by Daimler AG, fostering reproducibility [5]. The dataset contains 120 natural language requirements for two typical automotive systems, namely ALS and ADAS.

The requirements of the ALS describe a set of system functions: The functionality that causes the vehicle's direction indicators to flash in response to the steering column lever and hazard warning flasher switch. A function to lower the beams depending on the rotary light switch position and the vehicle setting for daytime running light. An adaptive high beam to control the high beam headlamp depending on the high beam switch and the detection of oncoming vehicles.

For the ADAS system, the dataset contains requirements concerning the main components for adaptive cruise control which maintains the distance to the vehicle in front and a speed set either manually by the driver or via traffic sign detection, provides a distance warning and an emergency brake assistant which reacts to stationary obstacles and to moving obstacles.

[1] https://arankomatsuzaki.wordpress.com/2021/06/04/gpt-j/

3 Related Work

NoRBERT [16] is a fine-tuned version of BERT [9] for deep learning-based requirements classification. It was trained on the PROMISE NFR dataset [8]. An approach for clustering natural language requirements is presented in [17]. The approach applies clustering to natural language descriptions with the idea of developing a DSL in mind. Apart from requirement classification, machine-learning and Natural Language Processing (NLP) can be used for prioritizing requirements [26]. These approaches however are not tailored for reducing the *initial* modeling efforts necessary for introducing Model-Driven Requirements Engineering (MDRE).

Approaches focusing on few-shot learning in requirements engineering exist, as well. One of these approaches uses transformer models for a named entity-recognition task [25]. Another related approach is a preliminary study on requirement classification using zero-shot learning [1]. The PROMISE NFR dataset is used with pre-trained Transformer models such as BERT and RoBERTa. Since it is in a preliminary status, only a reduced part of the dataset is used.

Apart from requirement classification, few-shot learning on requirements is also applicable for requirement elicitation as described in [29]. In contrast to other approaches in the area of few-shot learning in requirements engineering, the approach does not take already existing requirements as an input, but chat messages from which requirements for new hidden features are extracted.

4 Example DSL for Structured Requirements

The aim of this section is to design an exemplary requirements DSL increasing the degree of formality of requirement documents enabling automated verification and consistency checks. Then, we employ few-shot learning capabilities of large neural language models in order to transform existing requirements into the new syntax or to support the formulation of new requirements in an editor. In this work we focus on the automotive domain, and an existing set of ADAS and ALS requirements from [5].

We derived the following non-exhaustive set of requirements on DSLs that facilitate introducing MDRE in the automotive domain by reducing the training and translation efforts. **R1:** Since the modelers will not necessarily have a computer-science background, the DSL's syntax shall be based on natural language. **R2:** To make the language as intuitive as possible for its users and to enable the application of few-shot learning to implement an automatic translation from natural language requirements to models, the DSL's syntax shall be as close to the phrasing of requirements in natural language used by the modelers as possible. **R3:** In the DSL, requirements shall be formulated consistently and with a precise meaning understood by relevant stakeholders enabling automatic interpretation.

To engineer a DSL that fulfills the requirements R1-R3, the DSL designer must analyze and understand how the requirements are currently phrased and which meanings are implied by certain formulations.

The DSL is to be used in a natural language domain (*cf.* R1) and must ensure an intuitive readability for the requirements engineer as well as the modeler who is implementing the requirements at a later stage in the development process, *cf.* R3. The developed DSL follows an open-world assumption [12, 13], *i.e.,* whatever is not restricted by the model is allowed. Currently, the DSL focuses on isolated requirements written in one sentence. The reason for this is that we are mainly interested in concrete syntax and features such as unambiguity. Complex semantic connections and cross-references will not be discussed here.

A manual analysis of the requirements from [5] reveals a set of ambiguous or unstructured formulations and inconsistencies which might lead to misinterpretations and hence need to be tackled by the DSL approach. The following three ambiguity types are a non-exhaustive selection which is sufficient as a basis for our few-shot learning experiments.

Ambiguity 1: If-Then Constructs. The If-Then style is an often reoccurring pattern in requirements documents and an often occurring pattern. It reflects the idea that upon the occurrence of a trigger event something must happen. In standard English there are lots of variants how to express this, making it difficult for requirements engineers to stick to a consistent scheme, to search for such requirements, and to analyze them automatically. To tackle this difficulty, the first construct we introduce is the If-Then pattern. It not only creates clarity for the different stakeholders, cf. R3, but also makes further processing in trigger action patterns much easier [3]. Therefore, the DSL introduces the two keywords *IF:* and *THEN:*. For this purpose, the individual requirements are divided into two parts, a *trigger* part beginning with the keyword *IF:* and an *action* part starting with the keyword *THEN:*, i.e. a parsing rule is given as `if-then-req = 'IF:' trigger 'THEN:' action`. For the sake of simplicity we assume that the non-terminals `trigger` and `action` can be arbitrary strings.

This way we achieve a partial formalization of the original requirement. The trigger and the action are now clearly separated and the requirement can be identified as an If-Then requirement easily.

Ambiguity 2: Modal Verbs. The modal verbs (`must, can, should, etc.`) are important for the precise interpretation of requirements [27]. Moreover, in safety-critical systems such as vehicles, there is an important distinction between the available modal verbs. For instance, the modal verb *must* indicates a legal regulation and non-conformity can lead to legal consequences.

Our analysis shows however that the model verb is sometimes skipped. In such cases it might be not clear whether the given sentence is a requirement or a description of an

existing system. To enforce the usage of modal verbs we therefore introduce the dedicated keyword *MUST* in the DSL. In cases of missing modal verbs the keyword *MUST* needs to be included at the correct position. If a wrong modal verb such as *"can"* or *"could"* is used, it needs to be exchange by `MUST` thereby preventing the usage of weak words [27]. Requirements written without a modal verb should also be supplemented appropriately.

Ambiguity 3: Expressions. In requirements, we often need to quantify and compare things. Again, natural language offers many ways to describe comparisons, making it difficult to grasp the information of requirements automatically. For this reason, we introduce a third DSL construct for our DSL, namely logical formulae. Thereby, we are going to allow both: words (to keep the language close to natural formulations) and mathematical expressions in the DSL. For instance, we use the operators >= and <= for greater equal and less or equal. As an alternative we introduce the keywords *"GREATER"*, *"LESS"*, *"EQUAL"*, *"LESS EQUAL"*, and *"GREATER EQUAL"*. Such DSL constructs standardize the way how comparisons are formulated in a requirement. This facilitates automated consistency analysis since the operators and their semantics are clearly defined and the variables and constants can be extracted easily from the sentence. Now, if the same variable is used in another requirement, we can 1) link these requirements as treating the same context, e.g. to enable semantic requirement search and 2) are able to check whether the two requirements are consistent.

The three constructs introduced above should be combined when appropriate. For instance, the requirement *"The vehicle's doors are closed automatically when speeding velocity* ***is bigger than*** *10 km/h"* needs to be translated to *"****IF:*** *speeding velocity is* ***GREATER*** *10 km/h,* ***THEN:*** *the vehicle's doors* ***MUST*** *be closed automatically."*

The DSL requirement has almost no degrees of freedom in terms of formulation making it easy to extract the trigger variable (speeding velocity), the subject of the action (the vehicle's doors), and the desired state (closed automatically).

5 Translating Requirements to DSL

To automatically translate legacy natural language requirements into the DSL defined in Section 4, fixing bad formulations and enforcing guidelines usage and regulatory compliance, we utilize GPT-based language models. Since we need to avoid data and resource intensive finetuning (as the necessary amounts of data might lack in project or company-specific design and the required hardware resources might not be accessible/too expensive), we will make use of the few-shot capability, which has been shown to yield good results with only a few training examples.

The number of training examples for few-shot learning a new task is usually constrained by the context window, typically allowing 10-100 examples [6]. Sometimes the number of examples is further constrained by the available computational resources. To exploit the available training data as far as possible, we propose a cascaded translation process, where we provide a dedicated few-shot model for each translation task, i.e. trigger-action, modal verbs, and expressions. We expect this to reduce the few-shot complexity and to yield more focused models. The dedicated models need to be applied sequentially to incorporate all DSL constructs into a given requirement.

For each translation step, a set of few-shot examples, also referred to as the support set, for the respective requirements category is selected and given to the language model as context. Our hypothesis is that a large capacity language model pretrained on a sufficiently large training set can be used to solve a specific task such as a reformulation of a given requirement into a systematic form with a very small support set and without adapting the network's weights. The few-shot examples consist of input/output pairs and are input into the network as a demonstration for the task to be solved, followed by the actual query. The solution to the query is then generated as the model's output based on the support examples from the context.

The translation model can be implemented using any large enough pretrained language model supporting few-shot learning. Models of higher capacity can be expected to perform better in few-shot learned downstream tasks. Based on promising preliminary results, we decided to concentrate mainly on GPT-J-6B.

6 Evaluation

With the experiments conducted for the translation of requirements from unstructured text to DSL our aim was to find answers to the following research questions:

RQ1: Can state-of-the-art language models be employed to translate natural text requirements to systematic formulations based on few-shot learning?

RQ2: How many few-shot learning examples are required to train a translation rule?

To answer RQ1, we applied few-shot learning separately for If-Then requirements, modal verb insertion, and expressions (recall that for training each rule we use a separate instance of the language model). Input for the few-shot learning were pairs containing the unstructured input and the desired DSL formulation. To evaluate the "trained" language model, it had to transform an unstructured requirement that the model was not given as example into a requirement in the DSL. We then assessed the result of this transformation. We propose a custom evaluation scale with six possible quality classes. **Class 1**: The translation is both syntactically and semantically correct and fulfills the required formulation rule. No changes required. **Class 2**: The translation is semantically correct, but contains one or two syntactical inaccuracies to fully implement the desired rule. **Class**

Table 1. Evaluation results for the translation experiments from natural language requirements to domain-specific syntax.

# of Training Set	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	total
Translation results for If-Then structure using GPT-J6B							
1	2	0	1	0	6	2	11
4	5	2	0	1	1	2	11
6	6	3	0	0	0	2	11
Translation results for modal verbs structure using GPT-J6B							
1	3	0	0	0	5	0	8
4	5	0	0	0	3	0	8
6	6	0	0	0	2	0	8
Translation results for propositional logic Structure using GPT-J6B							
1 (trained on keyword: equal)	2	0	1	0	4	1	8
1 (trained on keyword: less or equal)	0	0	4	0	4	0	8
4	1	0	1	0	3	3	8
6	2	0	3	0	2	1	8
9	3	0	0	0	3	2	8

3: Syntactically correct but fails to fully cover the semantics of the source requirement (e.g. by missing a quantifier or a marginal constraint). **Class 4**: The translation contains one or two syntactical inaccuracies to fully implement the desired rule and the semantics is not fully covered, i.e. a combination of 2 and 3. **Class 5**: The translation has grave syntactical errors or does not implement the desired rule. An identity mapping would result in this label, as well (unless the input already implements the desired rule). **Class 6**: The translation is semantically wrong.

A flaw of this scale is that it is not ordinal. However, based on the experience we gathered with it in this work, in most cases a smaller number indicates a more satisfying result.

To answer RQ2 we conducted our evaluation with three differently sized few-shot support sets per translation rule consisting of one, four, and six examples each. In case of one-shot learning, *i.e.,* if only one example is presented in training, for the conversion of constraints containing (in)equalities, the result depends on the keyword used in the example. For this reason, we tried two different one-shot trainings, *i.e.,* for introducing "*EQUAL*", and "*LESS OR EQUAL*". The requirements used for testing were **not** present in the support sets. For instance, to one-shot train the translation of a requirement to the desired If-Then syntax we use the following input and output pair:
"Input: If a defective illuminant is detected, the information about the defective illuminant is transmitted to the instrument cluster.
DSL: IF: defective illuminant is detected, THEN: information about the defective illuminant is transmitted to the instrument cluster."

As we can see in the example, the two keywords *"IF:"* and *"THEN:"* are included in the target sentence. Apart from that the sentence remains almost unchanged, making it relatively easy for the model to learn the rule. Having seen this example, the model is already able to apply the rule perfectly to some examples of the test set, yielding a class 1 rating according to the scheme given above. For instance, the requirement

"If tip-blinking was activated shortly before deactivation of the hazard warning, this is not considered during the deactivation of the hazard warning." is correctly translated to *"IF: tip-blinking was activated shortly before deactivation of the hazard warning, THEN: this is not considered during the deactivation of the hazard warning ".*

However, some other examples are translated incompletely or wrong, e.g. the requirement *"With activated darkness switch (only armored vehicles) the cornering light is not activated."* is translated to *"IF: darkness switch is activated, THEN: cornering light is not activated. "*

While the keywords are included at the correct position, the model drops the information in parentheses stating that the requirement only applies to armored vehicles. For this reason, we consider this translation as syntactically correct, but semantically incomplete, resulting in a class 3 rating.

The experiments reveal some drawbacks inherent to models such as GPT. The models use statistically learned sequences without understanding the semantics. For instance, our model often confused less-than and greater-than inequalities due to the syntactic similarities. An overview of all experiment results is summarized in Table 1. As expected, in each of the three experiments, the translation quality improved with larger support sets. It is fascinating however, how steep the learning curve is. It suggests that few-shot learning can deal with natural language processing (NLP) tasks in requirements engineering even when only small training sets are available.

7 Conclusion

In this paper we have shown how neural language models such as representatives of the GPT family can support requirements engineering without the need for resource and data intensive fine-tuning. Our most important result is that few-shot learning of language models can be applied to translate legacy requirements into a given structured DSL form automatically. However, language models available today still require human supervision.

References

[1] Waad Alhoshan, Liping Zhao, Alessio Ferrari, and Keletso J. Letsholo. 2022. A Zero-Shot Learning Approach to Classifying Requirements: A Preliminary Study. In *Requirements Engineering: Foundation for Software Quality*, Vincenzo Gervasi and Andreas Vogelsang (Eds.). Springer International Publishing, Cham, 52–59.

[2] Abdallah Atouani, Jörg Christian Kirchhof, Evgeny Kusmenko, and Bernhard Rumpe. 2021. Artifact and Reference Models for Generative Machine Learning Frameworks and Build Systems. In *Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE '21)*. ACM SIGPLAN, 55–68.

[3] Jan Steffen Becker, Vincent Bertram, Tom Bienmüller, Udo Brockmeyer, Heiko Dörr, Thomas Peikenkamp, and Tino Teige. 2018. Interoperable Toolchain for Requirements-Driven Model-Based Development. In *9th European Congress Embedded Real Time Software and Systems ERTS '2018*.

[4] Vincent Bertram, Miriam Boß, Evgeny Kusmenko, Imke Helene Nachmann, Bernhard Rumpe, Danilo Trotta, and Louis Wachtmeister. 2022. Technical Report on Neural Language Models and Few-Shot Learning for Systematic Requirements Processing in MDSE. https://doi.org/10.48550/ARXIV.2211.09084

[5] Vincent Bertram, Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe, and Michael von Wenckstern. 2017. Component and Connector Views in Practice: An Experience Report. In *MODELS'17*.

[6] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[7] Antonio Buchhiarone, Jordi Cabot, Richard F. Paige, and Alfonso Pierantonio. 2020. Grand Challenges in Model-Driven Engineering: An Analysis of the State of Research. *Software and Systems Modeling* 19, 1 (2020).

[8] Jane Cleland-Huang, Sepideh Mazrouee, Huang Liguo, and Dan Port. 2007. *nfr*. https://doi.org/10.5281/zenodo.268542

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] Imke Drave, Akradii Gerasimov, Judith Michael, Lukas Netz, Bernhard Rumpe, and Simon Varga. 2021. A Methodology for Retrofitting Generative Aspects in Existing Applications. *Journal of Object Technology* 20 (November 2021), 1–24. https://doi.org/10.5381/jot.2021.20.2.a7

[11] I. Drave, T. Greifenberg, S. Hillemacher, S. Kriebel, E. Kusmenko, M. Markthaler, P. Orth, K. S. Salman, J. Richenhagen, B. Rumpe, C. Schulze, M. Wenckstern, and A. Wortmann. 2019. SMArDT modeling for automotive software testing. *Software: Practice and Experience* 49, 2 (February 2019), 301–328.

[12] Imke Drave, Timo Henrich, Katrin Hölldobler, Oliver Kautz, Judith Michael, and Bernhard Rumpe. 2020. Modellierung, Verifikation und Synthese von validen Planungszuständen für Fernsehausstrahlungen. In *Modellierung 2020*.

[13] Imke Drave, Oliver Kautz, Judith Michael, and Bernhard Rumpe. 2019. Semantic Evolution Analysis of Feature Models. In *SPLC'19* (Paris). ACM, 245–255.

[14] Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering (FOSE '07)* (May 2007), 37–54.

[15] David Harel and Bernhard Rumpe. 2004. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer* 37, 10 (2004).

[16] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F. Tichy. 2020. NoRBERT: Transfer learning for requirements classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 169–179.

[17] Katharina Juhnke, Alexander Nikic, and Matthias Tichy. 2021. Clustering Natural Language Test Case Instructions as Input for Deriving Automotive Testing DSLs. *The Journal of Object Technology* 20, 3 (2021).

[18] Nils Kaminski, Evgeny Kusmenko, and Bernhard Rumpe. 2019. Modeling Dynamic Architectures of Self-Adaptive Cooperative Systems. *The Journal of Object Technology* 18, 2 (July 2019), 1–20. https://doi.org/10.5381/jot.2019.18.2.a2 The 15th European Conference on Modelling Foundations and Applications.

[19] Hendrik Kausch, Mathias Pfeiffer, Deni Raco, and Bernhard Rumpe. 2020. An Approach for Logic-based Knowledge Representation and Automated Reasoning over Underspecification and Refinement in Safety-Critical Cyber-Physical Systems. In *Combined Proceedings of the Workshops at Software Engineering 2020* (Innsbruck), Vol. 2581. CEUR Workshop Proceedings.

[20] Evgeny Kusmenko, Sebastian Nickels, Svetlana Pavlitskaya, Bernhard Rumpe, and Thomas Timmermanns. 2019. Modeling and Training of Neural Processing Systems. In *Conference on Model Driven Engineering Languages and Systems (MODELS'19)* (Munich). IEEE, 283–293.

[21] Evgeny Kusmenko, Svetlana Pavlitskaya, Bernhard Rumpe, and Sebastian Stüber. 2019. On the Engineering of AI-Powered Systems. In *ASE'19. Software Engineering Intelligence Workshop (SEI'19)* (San Diego, California, USA), Lisa O'Conner (Ed.). IEEE, 126–133.

[22] Evgeny Kusmenko, Alexander Roth, Bernhard Rumpe, and Michael von Wenckstern. 2017. Modeling Architectures of Cyber-Physical Systems. In *ECMFA'17* (Marburg) *(LNCS 10376)*. Springer, 34–50.

[23] Grischa Liebel, Matthias Tichy, and Eric Knauss. 2019. Use, potential, and showstoppers of models in automotive requirements engineering. (2019).

[24] Grzegorz Loniewski, Emilio Insfran, and Silvia Abrahão. 2010. A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development. In *Model Driven Engineering Languages and Systems. MODELS'2010*.

[25] Garima Malik, Mucahit Cevik, Swayami Bera, Savas Yildirim, Devang Parikh, and Ayse Basar. 2022. Software requirement-specific entity extraction using transformer models. In *The 35th Canadian Conference on Artificial Intelligence*.

[26] Anna Perini, Angelo Susi, and Paolo Avesani. 2013. A Machine Learning Approach to Software Requirements Prioritization. *IEEE Transactions on Software Engineering* 39, 4 (2013).

[27] Klaus Pohl and Chris Rupp. 2021. *Basiswissen requirements engineering: Aus-und Weiterbildung nach IREB-Standard zum certified professional for requirements engineering foundation level*.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[29] Lin Shi, Mingzhe Xing, Mingyang Li, Yawen Wang, Shoubin Li, and Qing Wang. 2020. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. 641–653.

[30] Amit Kumar Tyagi and N. Sreenath. 2021. Cyber Physical Systems: Analyses, challenges and possible solutions. *Internet of Things and Cyber-Physical Systems* 1 (2021).

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[32] Ben Wang. 2021. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. https://github.com/kingoflolz/mesh-transformer-jax.

[33] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Received 2022-08-08; accepted 2022-09-30